Python – Class III

Identifiers, Reserved Words, DataTypes

LavaTech Technology

Agenda

- Identifiers
  - Rules for identifiers
- Reserved Keywords
  - 33 reserved words in detail
  - Rules for reserved keywords
- 14 Data Types
  - int
  - float
  - complex
  - bool
  - str
  - bytes
  - bytearray
  - range
  - list
  - tuple
  - set
  - frozenset
  - dict
  - None

Identifiers

What is an identifier ?

A Python identifier is a name of a variable, function, class, module or other object.

- Example -

  a1=50 ←Variable name is “a1”

  def addition(): ←Variable name is “addition”

  class Demo(): ←Variable name is “Demo”

Rules for identifiers

- Alphabets (upper case and lower case) and Digits (0 to 9) allowed.
- Underscore is allowed (_)
- Any other symbol apart from “_” is not allowed.
- Identifiers are case sensitive.
- Identifier should always start with alphabets and not digits.
- Keywords cannot be used as identifiers.
- There is no limit for length of python identifier.

- Valid Identifiers

  abc

  _total

  S1U1M

  var_name

  ANS

  _one_two

  __a__b

- Invalid Identifiers

  1abc

  if

  for

  an$

  &ns

  class

  work#one

- Note

  - Identifier starting with “_” are private.

  - Identifier starting with “__” are strongly private.

  - Identifier starting with “__” and ending with “__” like “__test__” , these are special variable defined by python

Reserved Words

What are Reserved words ?

- Keywords are the reserved words in Python. They are used to define the syntax and structure of the Python language.
- We cannot use a keyword as variable name, function name or any other identifier.
- In Python, keywords are case sensitive.
- There are 33 keywords in Python 3.3. This number can vary slightly in course of time.
- All the keywords except True, False and None are in lowercase and they must be written as it is.

Keywords in Python programming language

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

List all Keywords from Python Shell

```
root@server:~# python3
Python 3.5.2 (default, Nov 12 2018, 13:43:14)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import keyword
>>> print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally
', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'w
hile', 'with', 'yield']
>>>
```

True, False ← Boolean Values
None ← Null value
if,else,elif ← Conditional Statements
and,not,or,is ← Operators
while,break,continue,for,in,yield,return ← Looping Structures
try,finally,except,raise ← Exception Handling
assert ← Debugging tool
def ← Create Functions/Methods
lamba ← Create anonymouse Funtions
as ← Create Alias
import,from ← Include Libraries
class ← Create Class
pass ← Do nothing
global, nonlocal ← Scope of variables
del ← delete functions, variables etc
with ← File Handling

Note – Python does not contain “switch”, “goto”, “do..while”, “do..until”, “catch”, “throw”, ”throws” , ”private” , ”public”,“static”

Data Types

Data Types

- Every value in Python has a datatype.
- Everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

But what are classes and Objects ?

Rice

Veggies

Water

Masala

Class

Biryani

Khichdi

Pulavo

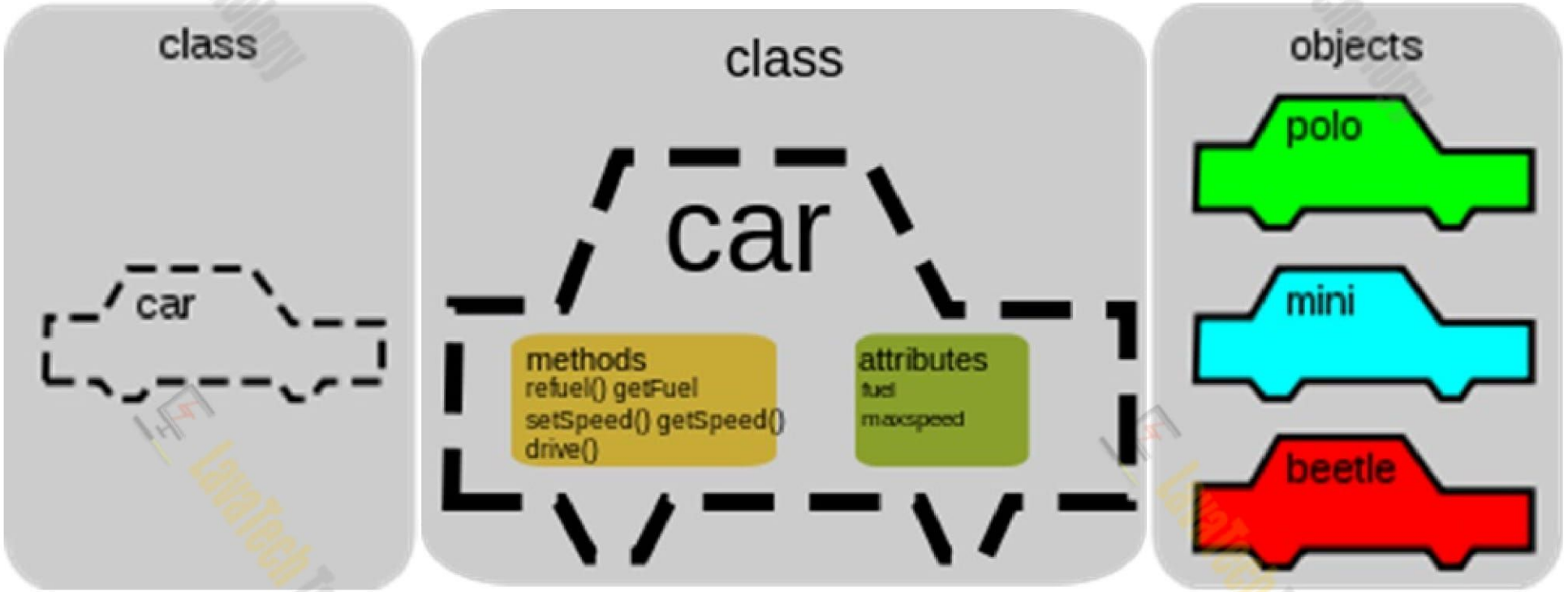
Masala Rice

Objects

What are Class and Objects?

- A class is a code template for creating objects.
- Object is collection of data (variables) and methods (functions). And, class is a blueprint for the object.
- We can think of class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house. House is the object.
- An object is also called an instance of a class and the process of creating this object is called instantiation.

Example



Python Numbers:
int , float , complex number

Integer Data Type - Int

- Used to represent WHOLE NUMBERS.
- Whole numbers do not have decimal points.
- There is no limit to how big the number can be.
- Upto Python2, there was long datatype, but long is not present in Python3.
- Integers can be negative, 0 or postive.

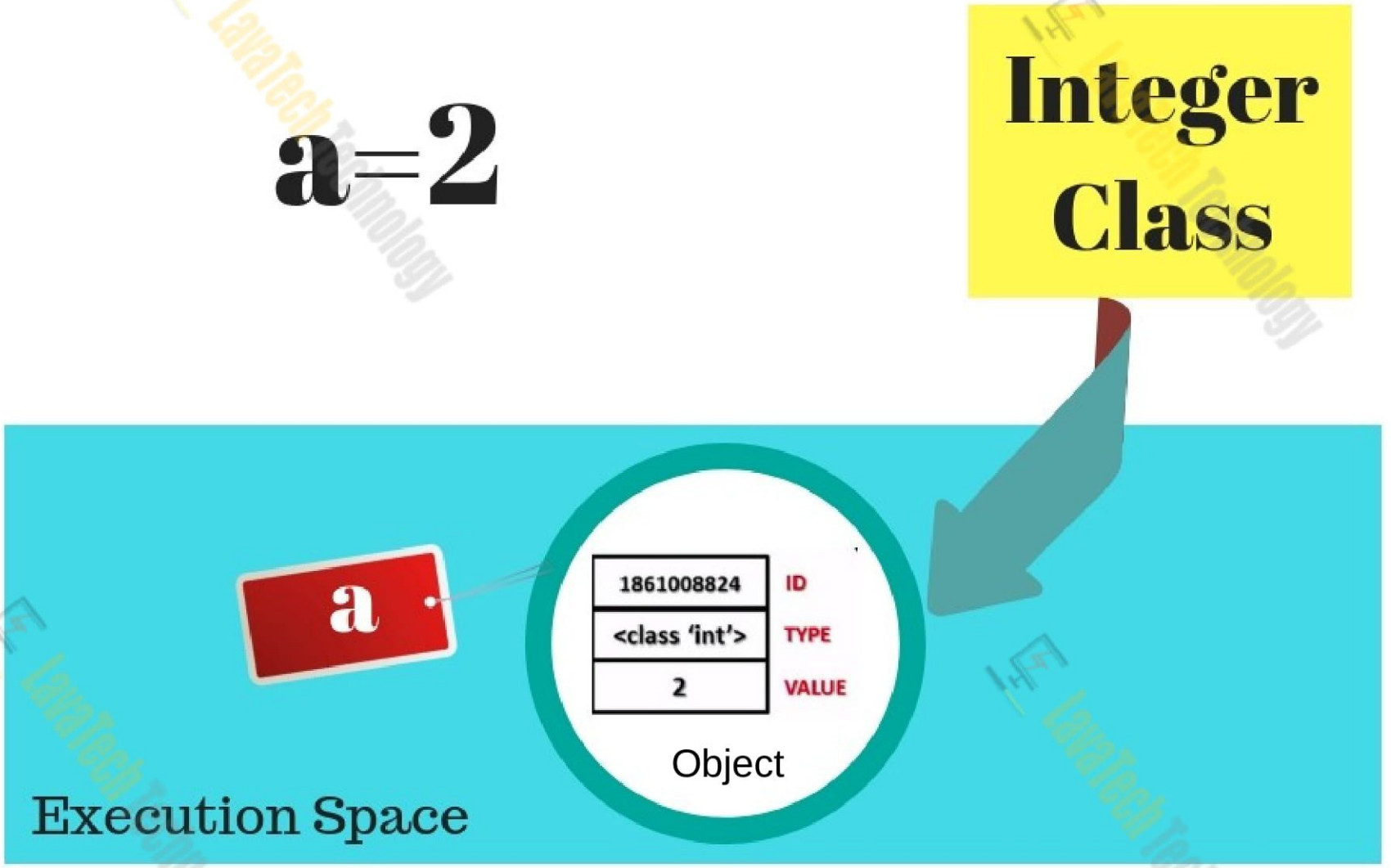
Example -

a=10 ← Without Declaring, we can start using

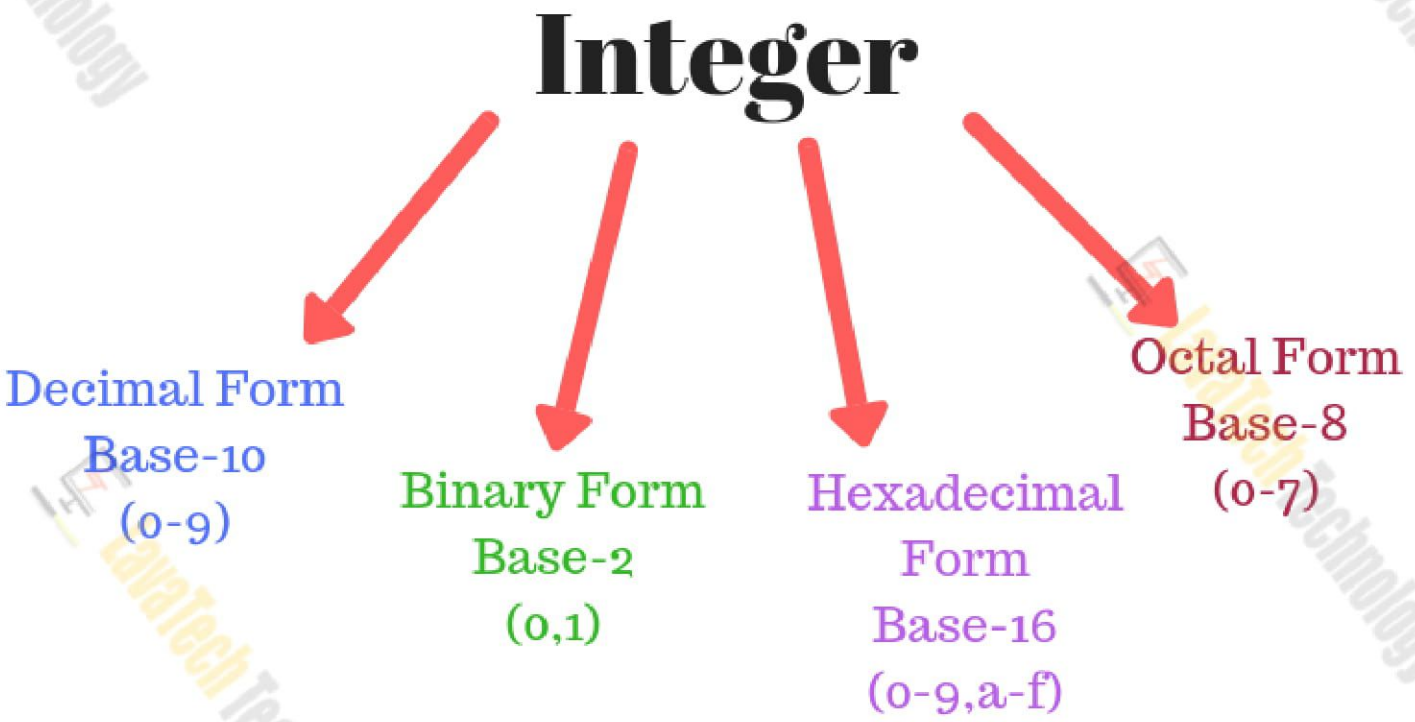
print(a) ← Display the content of “a”

type(a) ← Display the class of “a”

id(a) ← Memory Address of “a”



Ways to Represent Int Datatype



Binary Form

- Have only 2 charaters acceptable in binary form – 1 and 0. Hence it has Base 2
- In python, binary started with “0b” or “0B”
- Example -

  a=0000 ← Considered as “decimal”, not binary

  b=0b0000 ← Binary [Output of :print(b) → 0]

  c=0B11101 ← Binary [Output of: print(c) → 29]

  d=0b2134 ← Syntax error

  e=-0b101101 ← Binary [Output of: print(e) → -45]

- Notice that while printing, output is in decimal format!

Octal Form

- 8 charaters acceptable in octal form: 0 - 7. Hence it has Base 8.
- In python, Octal started with “0o” or “0O”
- Example -

  a=12345 ← Considered as “decimal”, not octal

  b=0o0010 ← Octal [Output of: print(b) → 8]

  c=0O1276 ← Octal [Output of: print(b) → 702]

  d=0o1289 ← Syntax error

  e=-0o101101 ← Octal [Output of: print(b) → -33345]

- Notice that while printing, output is in decimal format!

Hexadecimal Form

- 16 charaters acceptable in octal form: 0 to 9, a-f or A-F. Hence it has Base 16. This is one of few areas, where python is not case sensitive.
- In python, Octal started with “0x” or “0X”
- Example -

  a=12345 ← Considered as “decimal”, not hex

  b=0x0010 ← Hex [Output of: print(b) → 16]

  c=0xFa276 ← Hex [Output of: print(b) → 1024630]

  d=0X128g ← Syntax error

  e=-0X101101 ← Hex [Output of: print(b) → -1052929]

- Notice that while printing, output is in decimal format!

How to Print Actual Hex,Octal,Binary Values?
Or
How to Convert One form to another?

Using below functions:

1. bin() - Convert any form to binary

Example:
> bin(0X123)

> bin(12)

> bin(0O17)

```
>>>
>>> bin(0x1245)
'0b1001001000101'
>>>
>>> bin(0o17)
'0b1111'
>>>
>>>
>>> bin(19)
'0b10011'
>>>
>>>
>>> a=0xFFF
>>>
>>> bin(a)
'0b111111111111'
>>>
```

- Oct() - Convert any form to Octal

  Example -

  > oct(0X123)

  > oct(12)

  > oct(0b10101)

```
>>> oct(0xfa)
'0o372'
>>>
>>> oct(00165)
'0o165'
>>>
>>> oct(1245)
'0o2335'
>>>
>>> a=0xfc
>>>
>>> oct(a)
'0o374'
>>>
```

- hex() - Convert any form to hexadecimal

  Example -

  > hex(0b101011)

  > hex(12)

  > hex(0O17)

```
>>>
>>> hex(001267)
'0x2b7'
>>>
>>> hex(0b10101)
'0x15'
>>>
>>> hex(48)
'0x30'
>>>
>>> a=0b10101
>>>
>>> hex(a)
'0x15'
>>>
```

- int() - Convert binary,octal & hexadecimal to decimal -

  Example -

  > int(0b101011)

  > int(0xff)

  > int('0b1100',2)

  > int('0o123',8)

  > int('0xff',16)

```
>>>
>>> a=0xffaa11
>>> int(a)
16755217
>>>
>>> a=0o123
>>> int(a)
83
>>>
>>> a=0b10101
>>> int(a)
21
>>>
>>> a='0xffaa11'
>>> int(a,16)
16755217
>>>
>>> a='0o234'
>>> int(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '0o234'
>>>
>>> int(a,8)
156
>>>
>>> a='0b10101'
>>> int(a,2)
21
>>>
>>> a='\xffaa'
>>> int('ffaa',16)
65450
>>>
>>>
```

Practicle use case of Octal and Hex

- I want to print text with foreground and background color.
- User is entering information in octal or hexa-decimal format.

Pre-requite – Install PIP on windows using below link -

https://www.liquidweb.com/kb/install-pip-windows/

- Install Package named colr using pip -

On Windows -

pip install colr

On Linux -

pip3 install colr

Example 1: Using Functions we learnt

```
>>>
>>> from colr import color
>>>
>>> a=0o123412
>>>
>>> print(color('Welcome to Python Class',fore=oct(a)[2:],back='white'))
Welcome to Python Class
>>>
>>>
>>> b=0xff12a1
>>>
>>> print(color('Welcome to Python Class',fore=hex(b)[2:],back='white'))
Welcome to Python Class
>>>
>>> print(color('Welcome to Python Class',fore=hex(b)[2:],back='black'))
Welcome to Python Class
>>>
>>>
```

Example 2: If user is directly entering data in hex format, no need of functions -

```
>>>
>>> d=input("Enter hex:")
Enter hex:0fff12
>>>
>>> print(color('Welcome to Python Class',fore=d,back='black'))
Welcome to Python Class
>>>
>>>
```

Float - float

- Used to represent Number with decimal point.
- There is no limit to how big the number can be.
- Float numbers can be negative, 0 or postive.

  Example -

  a=10.0 ← Without Declaring, we can start using

  print(a) ← Display the content of “a”

  type(a) ← Display the class of “a”

  id(a) ← Memory Address of “a”

- Note – There is nothing like octal, binary or hexadecimal for for float

  Example -

  a=0x6.78 ← Syntax Error

  b=0b010.1 ← Syntax Error

- Exponential Form is allows for floating point data type -
- Example -

  a=1.2e100

```
>>> a=1.2e3
>>> print(a)
1200.0
>>>
>>> type(a)
<class 'float'>
>>>
>>> id(a)
140548712516728
>>>
```

What is need to Exponential Form?

Assume you have to record the population count of China or India from past 50 years!

Normal Integers size will be very big and accordingly the memory size utilized will be more.

With exponential form, big values can be stored in small space.

Example -

- To write 12 hundred crore -

  a=1.2e10

Complex Data type - Complex

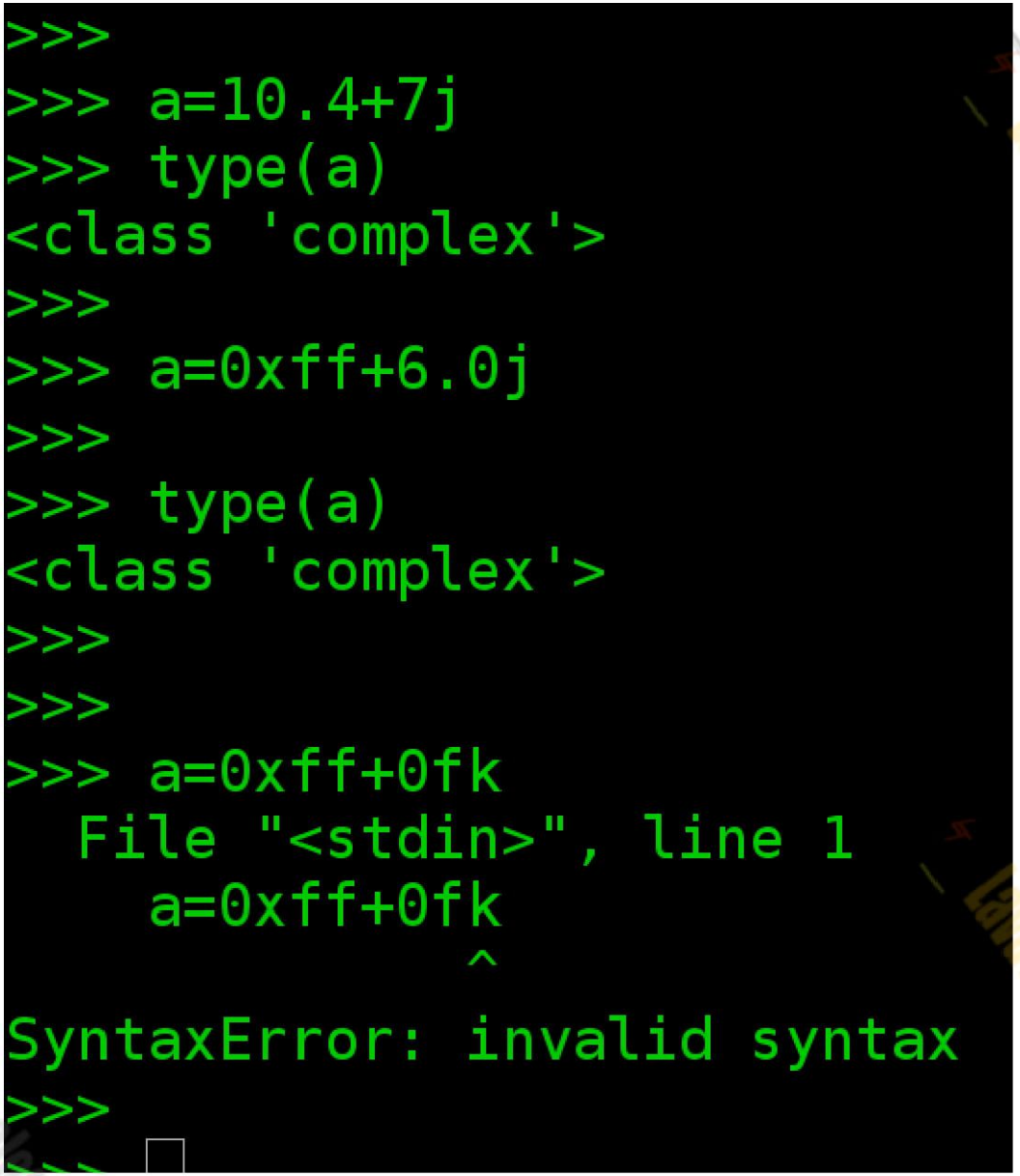
- Used for Mathematical and Scientific applications

  Normal Syntax of Complex number -

  a+bj where,

  a=real part

  b=imaginary part
- Real part & Imaginary part can be int, float.
- Imaginary part should be in demail format only and not in hexdecimal, octal or binary format.
- You cannot replace “j” with any other charater
- Order of “a+bj” can be “bj+a



Addition,Subraction,Multiplication,division can be performed for complex numbers

```
>>> a=10.4+7j
>>> b=5.2+6.1j
>>>
>>> a+b
(15.600000000000001+13.1j)
>>>
>>> a-b
(5.2+0.9000000000000004j)
>>>
>>> a*b
(11.380000000000001+99.84j)
>>>
>>> a/b
(1.5063035019455255-0.4208560311284047j)
>>>
```

Real and Imaginary numbers are stored as floating type internally

```
>>> a=0x12+7j
>>>
>>> a.real
18.0
>>>
>>> a.imag
7.0
>>>
>>> b=10+5j
>>>
>>> b.real
10.0
>>>
>>> b.imag
5.0
```

Boolean Data type - bool

- The Python type for storing true and false values is called bool.
- There are only two boolean values. They are True and False. Capitalization is important, since true and false are not boolean values (remember Python is case sensitive).

```
>>>
>>> print(True)
True
>>> print(true)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'true' is not defined
>>>
>>>
>>> print(type(True))
<class 'bool'>
>>> print(type(False))
<class 'bool'>
>>>
```

```
>>>
>>> a=true
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'true' is not defined
>>>
>>> a='True'
>>>
>>> type(a)
<class 'str'>
>>>
>>> a=True
>>>
>>> type(a)
<class 'bool'>
>>>
```

At memory level, True is stored as 1 and False is stored as 0

```
>>> True+True
2
>>> True+False
1
>>> False+False
0
>>> False+True
1
>>>
>>>
>>>
>>> a=10
>>> b=20
>>> a>b
False
>>> a<b
True
```

String Data Type: str

- Strings are sequences of character data. The string type in Python is called str.
- There is nothing like “char” in python.
- String literals may be delimited using either single or double quotes.

```
>>>
>>> print(I m a string)
  File "<stdin>", line 1
    print(I m a string)
            ^
SyntaxError: invalid syntax
>>>
>>> print("I m a string")
I m a string
>>>
>>> type('I m a string')
<class 'str'>
>>>
>>> a='example'
>>> type(a)
<class 'str'>
>>> a
'example'
>>>
```

Single Quote or Double Quote can be used to represent single line only and not multiple lines

To represent multiple line, we need to use 3 single/double quotes like -

```
'''hello
python'''
```

or

```
"""hello world
from python"""
```

- Suppose we want to print something like -

Welcome to “LavaTech Technology” for Python Class

Use 3 single/double quotes for this -

‘’’Welcome to “Lavatech Technology” for Python Class’’’

Or

“””Welcome to “Lavatech Technology” for Python Class”””

Consider below eg for difference between single quotes and triple quotes

```
root@server:~# cat format.py
a="""hi\n\thello"""
print(a)

b="""hi
        hello"""
print(b)

c="hi\n\thello"
print(c)

####### This will result in interpreter error #######
#d="hi
#       hello"
#print(d)
#####################################################

e="hi\\nhello"
print(e)

f="""hi\\nhello"""
print(f)

g="hi, I have lots of c@$h, do you want some?"
print(g)

h='''hi, I have lots of c@$h, do you want some?'''
print(h)
```

Escape Sequence in Python

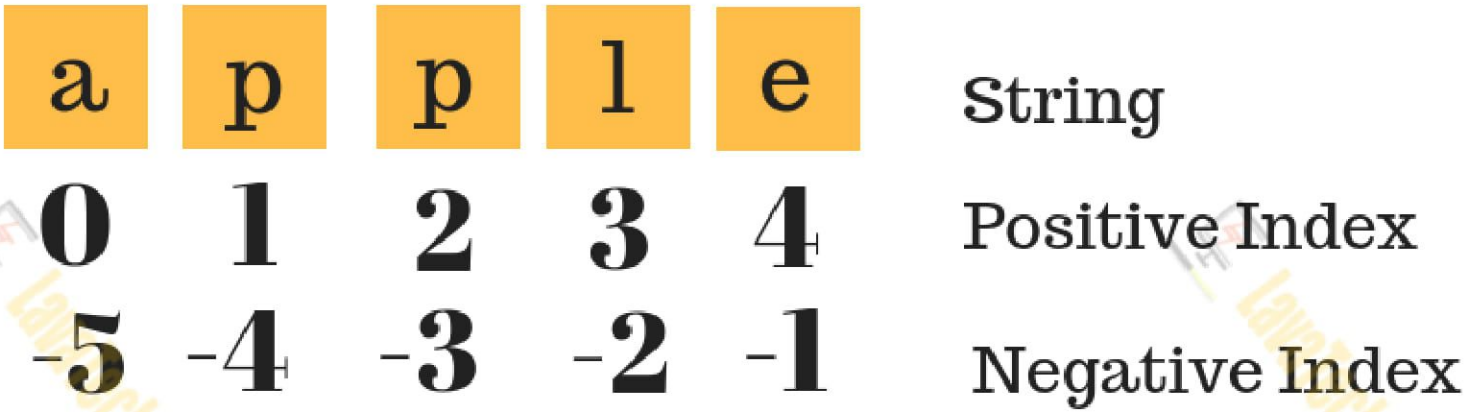
Escape Sequence	Description
\newline	Backslash and newline ignored
\\	Backslash
\'	Single quote
\"	Double quote
\a	ASCII Bell
\b	ASCII Backspace
\f	ASCII Formfeed
\n	ASCII Linefeed
\r	ASCII Carriage Return
\t	ASCII Horizontal Tab
\v	ASCII Vertical Tab
\ooo	Character with octal value ooo
\xHH	Character with hexadecimal value HH

Understanding Escape Sequence

```
root@server:~# cat format2.py
print("this is C:\Programs test ")
print("this is C:\\Programs test ")
print("this is C:\\\Programs test ")
print("this is /root/test test ")
print("this is /root//test test ")
print("this is /root///test test ")
######## Error ########
#print("Test to print "Lavatech Technology"")
############## ########
print("Test to print \"Lavatech Technology\"")
print("Test to print \'Lavatech Technology\'")
print("Test for\b\bbackspace")
print("Test for \f formfeed \f again")
print("Test for \nnewline")
print("Test for \rcarriage return")
print("Test for \thorizontal tab")
print("this is \v a test of \v vertical tab")
print("45 in octal means \45 \n456 in octal means \453")
print("This is \x48\x45\x58 representation\nff in hex representation is \xff")
root@server:~#
```

Understanding Index of String

- Individual string characters can be accessed using “index”
- In Python, index can be postive, 0 or negative
- Consider variable → a=”apple”



- Positive Index means → left to right
- Negative Index means → right to left
- Need of negative index is to provide flexiblilty to programmer.

```
>>>
>>> a="apple"
>>>
>>> a[0]
'a'
>>>
>>> a[2]
'p'
>>>
>>> a[3]
'l'
>>>
>>> a[-2]
'l'
>>>
>>> a[-4]
'p'
>>>
```

If wrong index is specified, you get → string index out of range error -

```
>>>
>>> a[-10]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>>
>>> a[100]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>>
```

Slice Operator

- A substring of a string is called a slice. Selecting a slice is similar to selecting a character
- Syntax of Slice Operator -
- var[begin:end]
- Note – Returns substring from begin to end-1

```
>>> a="apple"
>>>
>>> a[1:3]
'pp'
>>>
>>> a[0:4]
'appl'
```

- If you omit the first index (before the colon), the slice starts at the beginning of the string.
- If you omit the second index, the slice goes to the end of the string.
- The means, syntax is var[optional:optional]

```
>>>
>>> a="alphabet"
>>>
>>> a[:4]
'alph'
>>>
>>> a[0:]
'alphabet'
>>>
>>> a[3:]
'habet'
>>>
>>> a[:]
'alphabet'
>>>
```

- There is no Index Out Of Range exception for a slice. A slice is forgiving and shifts any offending index to something legal.

```
>>>
>>> a="alphabets"
>>>
>>> a[-4:100]
'bets'
>>>
>>> a[1:200]
'lphabets'
>>>
>>> a[-4:-100]
''
>>> a[-4:-6]
''
>>> a[-6:-4]
'ha'
>>> a[-100:-4]
'alpha'
>>>
```

Quiz

- str = 'lavatechtechnology'
- print('str = ', str)

- #first character
- print('str[0] = ', str[0])

- #last character
- print('str[-1] = ', str[-1])

- #slicing 2nd to 5th character
- print('str[1:5] = ', str[1:5])

- #slicing 6th to 2nd last character
- print('str[5:-2] = ', str[5:-2])

Understanding var[begin:end:step]

- Begin and end can be postive or negative, but step can be only positive.

```
>>>
>>> a="lavatechtechnology"
>>>
>>> a[0:19]
'lavatechtechnology'
>>>
>>> a[0:19:2]
'lvtctcnlg'
>>>
>>> a[0:19:-2]
''
>>> a[-19:-1:-2]
''
>>> a[-19:-1]
'lavatechtechnolog'
>>>
>>> a[-19:-1:-1]
''
>>> a[-19:-1:2]
'lvtctcnlg'
```

* operator with string

- To print a string multiple times, “*” operator is used.
- Syntax is -
- var*positive_integer

```
>>>
>>> a="lava"
>>>
>>> a*4
'lavalavalavalava'
>>>
>>> a*-4
''
>>> a*'3'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'str'
>>>
>>> a*0101
  File "<stdin>", line 1
    a*0101
         ^
SyntaxError: invalid token
```

Concatenation of Two or More Strings using “+” Operator

- Syntax -

  var1 + var2

```
>>> str1="hello"
>>> str2="world"
>>>
>>> print("Combo is:", str1+str2)
Combo is: helloworld
>>>
>>> print("Combo is:", str1+" "+str2)
Combo is: hello world
>>>
```

String Membership Test

- We can test if a sub string exists within a string or not, using the keyword in.

```
>>>
>>> a="strawberry"
>>>
>>> "raw" in a
True
>>>
>>> "berry" not in a
False
```

Finding len of string using → len()

- Syntax -

  len(var)

```
>>> a="lavatechtechnology"
>>>
>>> len(a)
18
>>>
>>> a[0:19]
'lavatechtechnology'
```

Python String Conversion Functions

Function Name	Description	Example Code
capitalize()	Returns the String with first character capitalized and rest of the characters in lower case.	var = 'PYTHON' print (var.capitalize()) # *Python*
lower()	Converts all the characters of the String to lowercase.	var = 'TechBeamers' print (var.lower()) # *techbeamers*
upper()	Converts all the characters of the String to uppercase.	var = 'TechBeamers' print (var.upper()) # *TECHBEAMERS*
swapcase()	Swaps the case of every character in the String means that lowercase characters are changed to uppercase and vice-versa.	var = 'TechBeamers' print (var.swapcase()) # *tECHbEAMERS*
title()	Returns the 'titlecased' version of String which means that all words start with uppercase and rest of the characters in the words are in lowercase.	var = 'welcome to Python programming' print (var.title()) # *Welcome To Python Programming*
count(str[,beg [,end]])	Returns the number of times substring 'str' occurs in range [beg, end] if beg and end index are given. If it is not given then substring is searched in whole String. Search is case-sensitive.	var='TechBeamers' str='e' print (var.count(str)) # *3* var1='Eagle Eyes' print (var1.count('e')) # *2* var2='Eagle Eyes' print (var2.count('E',0,5)) # *1*

Python String Comparison Functions, Section I

Function Name	Description	Example Code
islower()	Returns 'True' if all the characters in the String are in lowercase. If any one character is in uppercase it will return 'False'.	var='Python' print (var.islower()) # *False* var='python' print (var.islower()) # *True*
isupper()	Returns 'True' if all the characters in the String are in uppercase. If any one character is in lowercase it will return 'False'.	var='Python' print (var.isupper()) # *False* var='PYTHON' print (var.isupper()) # *True*
isdecimal()	Returns 'True' if all the characters in String are decimal. If anyone character in the String is of other data-type, it will return 'False'. Decimal characters are those from Unicode category 'Nd'. Complete list of 'Nd' is present at following link: http://www.fileformat.info/info/unicode/category/Nd/list.htm	num=u'2016' print (num.isdecimal()) # *True*
isdigit()	Returns 'True' for any character for which isdecimal() would return 'True and some characters in 'No' category. If there are any characters other than these, it will return 'False'. Precisely, digits are the characters for which Unicode property includes: Numeric_Type=Digit or Numeric_Type=Decimal. For example, superscripts are digits but fractions not.	print ('2'.isdigit()) # *True* print ('²'.isdigit()) # *True*

Python String Comparison Functions, Section II

Function Name	Description	Examples
isnumeric()	Returns 'True' if all the characters of the Unicode String lie in any one of the category 'Nd','No' and 'Nl'. If there are any characters other than these, it will return 'False'. Precisely, Numeric characters are those for which Unicode property includes Numeric_Type=Digit, Numeric_Type=Decimal or Numeric_Type=Numeric. Complete list of 'Nl' is present at following link: http://www.fileformat.info/info/unicode/category/Nl/list.htm	num=u'2016' print (num.isnumeric()) # *True* num=u'year2016' print (num.isnumeric()) # *False*
isalpha()	Returns 'True' if String contains at least one character (non-empty String) and all the characters are alphabetic, 'False' otherwise.	print ('python'.isalpha()) # *True* print ('python3'.isalpha()) # *False*
isalnum()	Returns 'True' if String contains at least one character (non-empty String) and all the characters are either alphabetic or decimal digits, 'False' otherwise.	print ('python'.isalnum()) # *True* print ('python3'.isalnum()) # *True*

Python String Padding Functions

Function Name	Description	Examples
rjust(width[,fillchar])	Returns a padded version of String with the original String right-justified to a total of width columns. By default, Padding is done by using space. Otherwise 'fillchar' specifies the filler character.	var='Python' print (var.rjust(10)) # *Python* print (var.rjust(10,'-')) # *Python*—-
ljust(width[,fillchar])	Returns a padded version of String with the original String left-justified to a total of width columns. By default, Padding is done by using space. Otherwise 'fillchar' specifies the filler character.	var='Python' print (var.ljust(10)) # *Python* print (var.ljust(10,'-')) # *Python*—-
center(width[,fillchar])	Returns a padded version of String with the original String moved to center to a total of width columns. By default, Padding is done by using space. Otherwise 'fillchar' specifies the filler character.	var='Python' print (var.center(20)) # *Python* print (var.center(20,'*')) # *******Python*******
zfill(width)	Returns a padded version of String with the original String padded on the left with zeros so that total length of String becomes equal to width. If there is a leading sign (+/-) present in the String, then with this function padding is done after the sign, not before it.	var='Python' print (var.zfill(10)) # *0000Python* var='+Python' print (var.zfill(10)) # *+000Python*

Functions To Find A String In Python, Section1

Function Name	Description	Example Code
find(str [,i [,j]])	Searches for 'str' in complete String (if i and j not defined) or in a sub-string of String (if i and j are defined).This function returns the index if 'str' is found else returns '-1'. where, i=search starts from this index j=search ends at this index.	var="Tech Beamers" str="Beam" print (var.find(str)) # 5 var="Tech Beamers" str="Beam" print (var.find(str,4)) # 5 var="Tech Beamers" str="Beam" print (var.find(str,7)) # -1
index(str[,i [,j]])	This is same as 'find' method. The only difference is that it raises 'ValueError' exception if 'str' is not found.	var='Tech Beamers' str='Beam' print (var.index(str)) # 5 var='Tech Beamers' str='Beam' print (var.index(str,4)) # 5 var='Tech Beamers' str='Beam' print (var.index(str,7)) # ValueError: substring not found

Functions To Find A String In Python, Section2

Function Name	Description	Example Code
rfind(str[,i [,j]])	This is same as find() just that this function returns the last index where 'str' is found. If 'str' is not found it returns '-1'.	var='This is a good example' str='is' print (var.rfind(str,0,10)) # *5* print (var.rfind(str,10)) # *-1*
count(str[,i [,j]])	Returns the number of occurrences of substring 'str' in the String. Searches for 'str' in complete String (if i and j not defined) or in a sub-string of String (if i and j are defined). where, i=search starts from this index j=search ends at this index.	var='This is a good example' str='is' print (var.count(str)) # *2* print (var.count(str,4,10)) # *1*

Functions To Replace A String In Python, Section1

Function Name	Description	Example Code
replace(old,new[,count])	Replaces all the occurrences of substring 'old' with 'new' in the String. If 'count' is defined then only 'count' number of occurrences of 'old' will be replaced with 'new'. where, old =substring to be replaced new =substring that will replace the old count =number of occurrences of old that will be replaced with new.	var='This is a good example' str='was' print (var.replace('is',str)) # *Thwas was a good exampleprint (var.replace('is',str,1))* # *Thwas is a good example*
split([sep[,maxsplit]])	Returns a list of substring obtained after splitting the String with 'sep' as delimiter. where, sep= delimiter, default is space maxsplit= number of splits to be done	var = "This is a good example" print (var.split()) # *['This', 'is', 'a', 'good', 'example']print (var.split(' ', 3))* # *['This', 'is', 'a', 'good example']*

Functions To Replace A String In Python, Section2

Function Name	Description	Example Code
splitlines(num)	Splits the String at line breaks and returns the list after removing the line breaks. where, num = if this is positive value. It indicates that line breaks to be included in the returned list.	var='Print new line\nNextline\n\nMove again to new line' print (var.splitlines()) # *['Print new line', 'Nextline', '', 'Move again to new line']* print *(var.splitlines(1))* # *['Print new line\n', 'Nextline\n', '\n', 'Move again to new line']*
join(seq)	Returns a String obtained after concatenating the sequence 'seq' with a delimiter string. where, seq= sequence of elements to be joined	seq=('ab','bc','cd') str='=' print (str.join(seq)) # *ab=bc=cd*

Misc String Handling Functions In Python, Section1

Function Name	Description	Example Code
lstrip([chars])	Returns a String after removing the characters from the beginning of the String. where, Chars=this is the character to be trimmed from the String. Default is whitespace character.	var=' This is a good example ' print (var.lstrip()) # *This is a good example* var='*****This is a good example*****' print (var.lstrip('*')) # *This is a good example***********
rstrip()	Returns a String after removing the characters from the End of the String. where, Chars=this is the character to be trimmed from the String. Default is whitespace character.	var=' This is a good example ' print (var.rstrip()) # *This is a good example* var='*****This is a good example*****' print (var.lstrip('*')) # *****This is a good example*

Misc String Handling Functions In Python, Section2

Function Name	Description	Example Code
rindex(str[,i [,j]])	Searches for 'str' in complete String (if i and j not defined) or in a sub-string of String (if i and j are defined).This function returns the last index where 'str' is found. If 'str' is not found it raises 'ValueError' exception.where, i=search starts from this index j=search ends at this index.	var='This is a good example' str='is' print (var.rindex(str,0,10)) # *5* print (var.rindex(str,10)) # *ValueError: substring not found*
len(string)	Returns the length of given String	var='This is a good example' print (len(var)) # *22*

Activity - I

- User is entering name of image to be save
  - Append “png” at end of name entered : Hint→ “+”
  - Strip “jpeg” if user have entered that, and replace it with png : Hint → var.replace(“.jpeg”,”.png”)
- User is entering package name as – xorg-x11-drv-modesetting.x86_64.rpm
  - Strip only name of package : Hint → a=d.split(“.”) c=a[0]
  - Strip architecture of package : Hint → c[1]
  - Strip name and architecture of package : Hint → c[0]+”.”+c[1]
- Accept name and surname of user, create a folder with name conditions →
  - Capitalise the initial of name and surname : Hint → c=d.title()
  - Preserve only Initals of name and Surname : Hint → a=c[0][0]+c[1][0]
  - First-letter & last letter of name &surname : Hint → a=c[0][0]+c[0][len(c)+1]+c[1]
  - Only surname : Hint c[1]
  - First-letter & last letter of name repeated twice and surname : Hint → a=c[0][0]+c[0][len(c)+1] ; k=a*2+c[1]
  - Only name : Hint → c[0]

Activity - II

- User is entering mobile number as +91-9034129234
  - Strip “+91-” from the string, output should be 9034129234
  - Reformat the date as 919034129234
  - Remove “+91-” and add “0”, output should be 09034129234
- User is entering IP address of a system → 192.168.1.10
  - Format the IP address as → a-w-s-192-168-1-10
- User is entering birthdate at → 5Jan,2016
  - Format String to have output → 5 Jan 2016
  - Format String to have output → Jan 5, 2016
  - Format String to have output → 2016, Jan
- User is entering date in format – 10/10/2016
  - Format the date as 10-10-2016
  - Strip the “/” from date entered, output should be 10102016
  - Format the date as → Date-10,Month-10,Year-2016

Activity - III

Accept the details from user as -

- Enter Username :jack sparrow
- Enter PhoneNumber :+91-923102321202

Create a file and stored the details entered by user in below format -

Username is: jack sparrow

PhoneNumber is: +91-923102321202

Your Nick Name is: JS

Your Password is: JKsparrow

Code for Activity - III

```
[root@sandbox ~]#
[root@sandbox ~]# cat info.py
import os

f=open("test","w")
a=input("Enter Username :")
b=input("Enter PhoneNumber :")
nick_name=a.split(" ")[0][0].title()+a.split(" ")[1][0].title()
password=a.title().split(" ")[0][0]+a[a.rfind(" ")-1].title()+a.split(" ")[1]

f.write("Username is: "+ a + "\n")
f.write("PhoneNumber is: "+ b + "\n")
f.write("Your Nick Name is: "+nick_name+"\n")
f.write("Your Password is: "+password+"\n")
f.close()
```

- Python’s Fundamental Datatype are -
  - Int
  - Float
  - Complex
  - Bool
  - Str

Note – In python, char datatype is not present.

Type Casting also called Type Coersion

- The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion. Python has two types of type conversion.

  Implicit Type Conversion

  Explicit Type Conversion

Implicit Type Conversion:

In Implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement.

Let's see an example where Python promotes conversion of lower datatype (integer) to higher data type (float) to avoid data loss.

Example 1: Converting integer to float

```
root@server:~# cat casting1.py
### Start ###
num_int = 123
num_flo = 1.23

num_new = num_int + num_flo

print("datatype of num_int:",type(num_int))
print("datatype of num_flo:",type(num_flo))

print("Value of num_new:",num_new)
print("datatype of num_new:",type(num_new))
### End ###
root@server:~#
root@server:~#
root@server:~# python3 casting1.py
datatype of num_int: <class 'int'>
datatype of num_flo: <class 'float'>
Value of num_new: 124.23
datatype of num_new: <class 'float'>
```

- In the program,
  - In the output we can see the datatype of num_int is an integer, datatype of num_flo is a float.
  - Also, we can see the num_new has float data type because Python always converts smaller data type to larger data type to avoid the loss of data.

- Now, let's try adding a string and an integer, and see how Python treats it.

Example 2: Addition of string(higher) data type and integer(lower) datatype

```
root@server:~# cat casting2.py
### Start ###
num_int = 123
num_str = "456"

print("Data type of num_int:",type(num_int))
print("Data type of num_str:",type(num_str))

print(num_int+num_str)
### End ###
root@server:~#
root@server:~# python3 casting2.py
Data type of num_int: <class 'int'>
Data type of num_str: <class 'str'>
Traceback (most recent call last):
  File "casting2.py", line 8, in <module>
    print(num_int+num_str)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

- In the program,
  - As we can see from the output, we got typeerror. Python is not able use Implicit Conversion in such condition.

- However Python has the solution for this type of situation which is know as Explicit Conversion.

Explicit Type Conversion

- In Explicit Type Conversion, users convert the data type of an object to required data type.
- We use the predefined functions like int(), float(), complex(), bool(), str(), to perform explicit type conversion.
- This type conversion is also called typecasting because the user casts (change) the data type of the objects.
- Below functions are used for this -
  - int()
  - float()
  - str()
  - complex()
  - bool()

int()

- Int() function is used to convert any value which is in other datatype format but in base10 form to integer.
- If integer is of hexademical, octal , binary format, then too it will convert to integer
- Complex Number to Integer will also result in type error.
- Syntax -

  int(expression/variable)

```
>>> int(5.9)
5
>>> int(5+9j)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't convert complex to int
>>>
>>> int(True)
1
>>> int(False)
0
>>> int("10")
10
>>> int("10.5")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '10.
5'
>>>
```

float()

float() function is used to convert any value which is in other datatype format but in base10 form or floating number format to float. Complex Number to float will also result in type error

- Syntax -

  float(expression/variable)

```
>>> float(14)
14.0
>>> float(5+9j)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't convert complex to float
>>>
>>> float("10")
10.0
>>>
>>> float("100.99")
100.99
>>>
>>> float("six")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: 'six'
>>>
>>> float(0xffa12)
1047058.0
>>>
>>> float(0B1111)
15.0
```

complex()

- complex() function is used to convert other type to complex type.
- Syntax -

  complex(a) →will result in complex number - a+0j

  complex(a,b) →will result in complex number - a+bj

  - Here, a should be of integer, floating number, True or Flase
  - b should be in base10, base2, base16, base8 format
  - complex() can't take second arg if first is a string

```
>>> complex(10)
(10+0j)
>>>
>>> complex(100.99)
(100.99+0j)
>>>
>>> complex(True)
(1+0j)
>>>
>>> complex(False)
0j
>>>
>>> complex("five")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: complex() arg is a malformed string
>>>
>>> complex(0B101)
(5+0j)
>>>
>>> complex("10",6)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: complex() can't take second arg if first is
a string
>>>
>>> complex(0B101,7)
(5+7j)
```

bool()

bool() function is used to convert any value which is in other datatype format to boolean type.

This function will result into only 2 values → True or False, where True represents 1 and False represents 0

- Syntax -

  bool(expression/variable)

```
>>> bool(0)
False
>>>
>>> bool(1)
True
>>>
>>> bool(20)
True
>>>
>>> bool(5+6j)
True
>>>
>>> bool("Apple")
True
>>>
>>>
>>> bool(0B12)
  File "<stdin>", line 1
    bool(0B12)
             ^
SyntaxError: invalid syntax
>>>
>>> bool(0B10)
True
>>>
>>> bool('')
False
>>>
>>> bool(' ')
True
>>> bool(0.1)
True
```

str()

str() function is used to convert any type of value to string format

- Syntax -

  str(expression/variable)

```
>>> str(10+4j)
'(10+4j)'
>>>
>>> str(100.99)
'100.99'
>>>
>>> str(Lava)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Lava' is not defined
>>>
>>> str(True)
'True'
>>>
>>> str(False)
'False'
```

Activity

```
[root@sandbox ~]# cat type.py
a=input("Enter no1 :")
b=input("Enter no2 :")
print("Addition is:",int(a)+int(b))

############***Error***############
#  print("Addition is: "+ int(a)+int(b))
############***Error***############
print("Addition is: "+ str(int(a)+int(b)))
```

```
[root@sandbox ~]# cat type.py
a=input("Enter no1 :")
b=input("Enter no2 :")
print("Addition is:",int(a)+int(b))

############***Error***############
#  print("Addition is: "+ int(a)+int(b))
############***Error***############
print("Addition is: "+ str(int(a)+int(b)))
```